THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

July 25, 2003

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *60/390,576*
FILING DATE: *June 21, 2002*
RELATED PCT APPLICATION NUMBER: *PCT/US03/19674*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

P. R. Grant

P. R. GRANT
Certifying Officer

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

Please type a plus sign (+) inside this box → +

06-24-02

PTO/SB/16

Approved for use through 10/31/2002. OMB 0651-
U S Patent and Trademark Office, U.S DEPARTMENT OF COMME

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control num

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

06/20/02 JC784 U.S. PTO

JC978 U.S. PTO 60390576

CERTIFICATE OF MAILING BY "EXPRESS MAIL"

Express Mail Label No.: EV 093213306 US Date of Deposit: June 21, 2002

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231

Tamara Alcaraz

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	RESIDENCE (City and either State or Foreign Country)
Nicholas	AMATO	Ann Arbor, Michigan

☐ Additional inventors are being named on the ___ separately numbered sheets attached hereto.

TITLE OF THE INVENTION (280 characters max)

FIBONACCI HEAP FOR USE WITH INTERNET ROUTING PROTOCOLS

CORRESPONDENCE ADDRESS

Direct all correspondence to:

Place Customer Number Bar Code Label here

☒ Customer Number 25226

Type Customer Number here

25226

PATENT TRADEMARK OFFICE

OR

☐ Firm or Individual Name					
Address					
Country		Telephone		Fax	

ENCLOSED APPLICATION PARTS (check all that apply)

☒ Specification *Number of Pages (including figures)* 30	☐ CD(s), Number
☐ Drawing(s) *Number of Sheets*	☒ Other (specify) Return Postcard
☒ Application Data Sheet, See 37 CFR 1.76 - 2 pages	

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT

	FILING FEE AMOUNT($)
☐ Applicant claims small entity status. See 37 CFR 1.27.	
☐ A check or money order is enclosed to cover the filing fees.	
☒ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account No.: 03-1952.	$160.00
☐ Payment by credit card. Form PTO-2038 is attached.	

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are: _____

Respectfully submitted,

SIGNATURE _____

TYPED or PRINTED NAME: Alan S. Hodes

TELEPHONE: (650) 813-5622

Date: June 21, 2002

REGISTRATION NO. 38,185 (if appropriate)

Docket Number: 529223000100

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U S Patent and Trademark Office, U S Department of Commerce, Washington, D C 20231 DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS SEND TO: Box Provisional Application, Commissioner for Patents, Washington, D.C. 20231.

pa-704285

Application Data Sheet

Inventor Information

Inventor One Given Name:	Nicholas
Family Name:	AMATO
Name Suffix:	
Postal Address Line One	1198 Paddock Pl.
Postal Address Line Two	#107
City:	Ann Arbor
State or Province:	Michigan
Postal or Zip Code:	48108
Citizenship Country:	United States of America

Correspondence Information

Name Line One:	Alan S. Hodes
Name Line Two:	Morrison & Foerster LLP
Address Line One:	755 Page Mill Road
Address Line Two:	
City:	Palo Alto
State or Province:	California
Postal or Zip Code:	94304-1018
Telephone:	(650) 813-5657
Fax:	(650) 494-0792
Electronic Mail:	ahodes@mofo.com

Application Information

Title Line One:	FIBONACCI HEAP FOR USE WITH
Title Line Two:	INTERNET ROUTING PROTOCOLS
Total Drawing Sheets:	
Formal Drawings?:	
Application Type:	Provisional
Docket Number:	529223000100

Representative Information

Representative Customer Number:	25226

Continuity Information

This application is a:
> Application One:
Filing Date:

which is a:
>>Application Two:
Filing Date:

which is a:
>>>Application Three:
Filing Date:

Prior Foreign Applications

Foreign Application One:
Filing Date:
Country:
Priority Claimed:

FIBONACCI HEAP FOR USE WITH INTERNET ROUTING PROTOCOLS

PURPOSE

[0001] This document describes a general algorithm and data structure (specialized Fibonacci Heap) and, also, its particular application to OSPF and IS-IS routing protocols.

[0002] A specialized Fibonacci Heap implementation is suited for application to Internet routing protocols that use a Dijkstra-like algorithm to determine a shortest paths tree. Examples of such Internet routing protocols include OSPF and IS-IS.

DESCRIPTION OF INTERNET ROUTING PROTOCOLS

[0003] Information in the Internet is transmitted as *packets*. A packet in the Internet is a fixed-length piece of data that is individually routed hop-by-hop from source to destination. The action of *routing* a packet means that each router along the path examines header information in the packet and a local database in order to forward the packet to its next hop. This local database is typically called the *Forwarding Information Base* or FIB. Entries in the FIB, usually structured as a table, determine to where packets are forwarded. The FIB is derived from a collective database called a *Routing Information Database* or RIB. This RIB is a collection of all the routing information the router "knows"; an algorithm maps the entries (routes) in the RIB to those in the FIB, which is used for forwarding. The RIB is typically built in two ways, which may be used together: (a) static configuration, and (b) dynamic routing protocols. These protocols may be further subdivided into two groups based on the part of the Internet in which they operate: exterior gateway protocols, or EGPs, are responsible for the dissemination of routing data between autonomous administrative domains, and interior gateway protocols, or IGPs, are responsible for dissemination of routing data within a single autonomous domain. Furthermore, two types of IGPs are in widespread

use today: those that use a distance-vector type of algorithm and those that use the link-state method. This document addresses the application of an algorithm to optimize a computation performed in the operation of link-state IGPs.

Link State And The Dijkstra-Like Algorithm

[0004] An object represented by a link state routing protocol is either a multi-access network or a router. Networks may be connected to routers, but not to other networks. Routers may be connected to networks (by multi-access interfaces) or other routers (by different classes of point-to-point interfaces) Computer networks are a special case of an abstract mathematical structure called a *graph*. That is, the graph represents the topology of the network. Link state routing protocols allow routers to store an internal representation of the graph in a domain.

[0005] The graph includes *vertices* and *edges*, where vertices are either hosts (end systems that do not route packets not locally originated or destined) or routers (systems that may route packets to a "next hop"). Each edge connects a pair of vertices.

[0006] The IGP protocols define, at least broadly, four primary features:

- Operation of flooding link state information.
- Structure of link state information.
- Algorithm for computing a shortest path tree.
- Sub-protocols for neighbor acquisition and database synchronization, packet formats for communication.

[0007] Each router floods an "advertisement" describing its local connectivity. The protocol defines a flooding mechanism aimed at ensuring the data is transmitted throughout the domain, giving each participant the same view of the network. A standard algorithm is used to compute a shortest path tree on the resulting graph. This allows hop-by-hop routing to function, as all routers will have the same idea about what the shortest paths are in the network.

[0008] While these protocols operate on the abstract concept of a "graph," each protocol defines how the graph is represented and how to compute shortest

path trees. Thus, in general, the definition of how the graph is represented differs among the protocols. The OSPF and IS-IS protocols are described in this document as examples.

[0009] Many link-state routing protocols use a Dijkstra-like algorithm to compute shortest paths. These algorithms refer to an abstract structure called a *candidate list*, which contains nodes that have been visited in the computation but to which it is not known if the shortest paths have been discovered. The implementation of the candidate list depends in part on which specific protocol is used. The list contains routers and hosts (vertices) or networks (edges). As defined by the protocol standards, the "list" is simply a set of routers and networks, and the standards do not otherwise require any particular representation of the set.

Use Of A Fibonacci Heap In Link State Protocols

[0010] An abstract algorithm is used to optimize the Dijkstra-like algorithm to the candidate list of routers and networks described in Internet link state protocols. Each routing protocol uses a corresponding algorithm to compute a shortest path tree. The use of the Fibonacci heap greatly improves the speed of the computation, allowing the algorithm to be run more often and with fewer restrictions.

[0011] A generalized implementation of a Fibonacci heap is specially tailored for the algorithms used in Internet link state protocols. In OSPF and IS-IS, this is the candidate list used in Section 16.1 of RFC 2328 (OSPF) and the TENT list (IS-IS). Examples of special modifications are:

- An API designed to satisfy the needs of Internet link state protocols while representing the list as a Fibonacci heap. The API is tailored to the specific needs of the shortest path computation in OSPF and IS-IS, for example.
- Implementation of the algorithm's data structures so that the algorithm may operate on any link state protocol objects (routers and networks in OSPF and IS-IS, for example).
- Minimize or avoid recursion. Recursion is particularly disadvantageous on systems with limited stack space.

[redacted]

- Allocation of the "auxiliary" array at initialization time. The array is a fixed size, the maximum base-2 log of the largest path metric expected in a shortest path computation.

Use of a comparison function to increase the usability of the heap on different data structures.

API

[0012] The generalized API includes the following operations:

- Initialization
- Insertion
- Relax Key
- Extract Minimum

[0013] These operations are used in the computation of shortest path trees for the purpose of Internet routing. The API may operate on OSPF (Router and Network LSAs) or IS-IS (LSPs and Pseudonode LSPs) without code modifications; the API accepts generic descriptions of these structures (a "node") and operates on each in the same way, regardless of what the nodes represent.

[0014] This allows the API to be used for multiple purposes, for example, in both OSPF and IS-IS.

[0015] In the *initialization* operation, the maximum sized auxiliary array is allocated according to a parameter to the initialization function.

Generic Data Structures

[0016] An **fnode_t** structure, described in more detail later, is a general way of representing a piece of link state information: for example, an OSPF Router LSA or Network LSA. This structure contains information specific only to the Fibonacci heap.

[0017] The node may be offset or otherwise referenced into a protocol data structure, such as an LSA representation, by giving a value specifying an offset or other reference of the heap "key" to the initialization API call. This allows further operations to reference the key of the node without awareness of the protocol-specific data structures.

Minimization Or Elimination Of Recursion

[0018] A recursive implementation of this algorithm can be impractical due to the limited amount of stack space on many systems. This implementation uses an iterative version of the "cutting" component of the Fibonacci heap algorithm.

Comparison Function

[0019] A comparison function is stored in the Fibonacci heap instance structure. This function is called with two arguments, which are references to the keys of two nodes to be compared. The function returns 0, -1 or 1 to indicate the first key is equal to, lesser than or greater than the second key, respectively.

Purpose

[0020] As discussed above, Internet Routing protocols such as OSPF (Open Shortest Path First) utilize a *candidate list* to maintain a set of vertices that have been visited in the Dijkstra-like computation. The operations performed on the list may be summarized as *insert*, *extract-minimum*, and *relax-key*. Each vertex contains some key which is a numeric value, for example, representing the current best cost to that vertex from the source.
With the application of the specialized Fibonacci Heap structure, the Dijkstra-like algorithm used to compute shortest path trees runs with $O(V \lg V + E)$ complexity, **resulting in improved IP network scalability.**

Operation

[0021] A specialized algorithm and data structure are applied to the representation of a critical piece of the Dijkstra-like algorithm: the candidate list. The application of this algorithm and data structure results in increased scalability in link state Internet routing protocols by making the shortest path (Dijkstra-like) computation more efficient.

[0022] **Figure 1** shows an example of the functional placement of the algorithm and data structure.

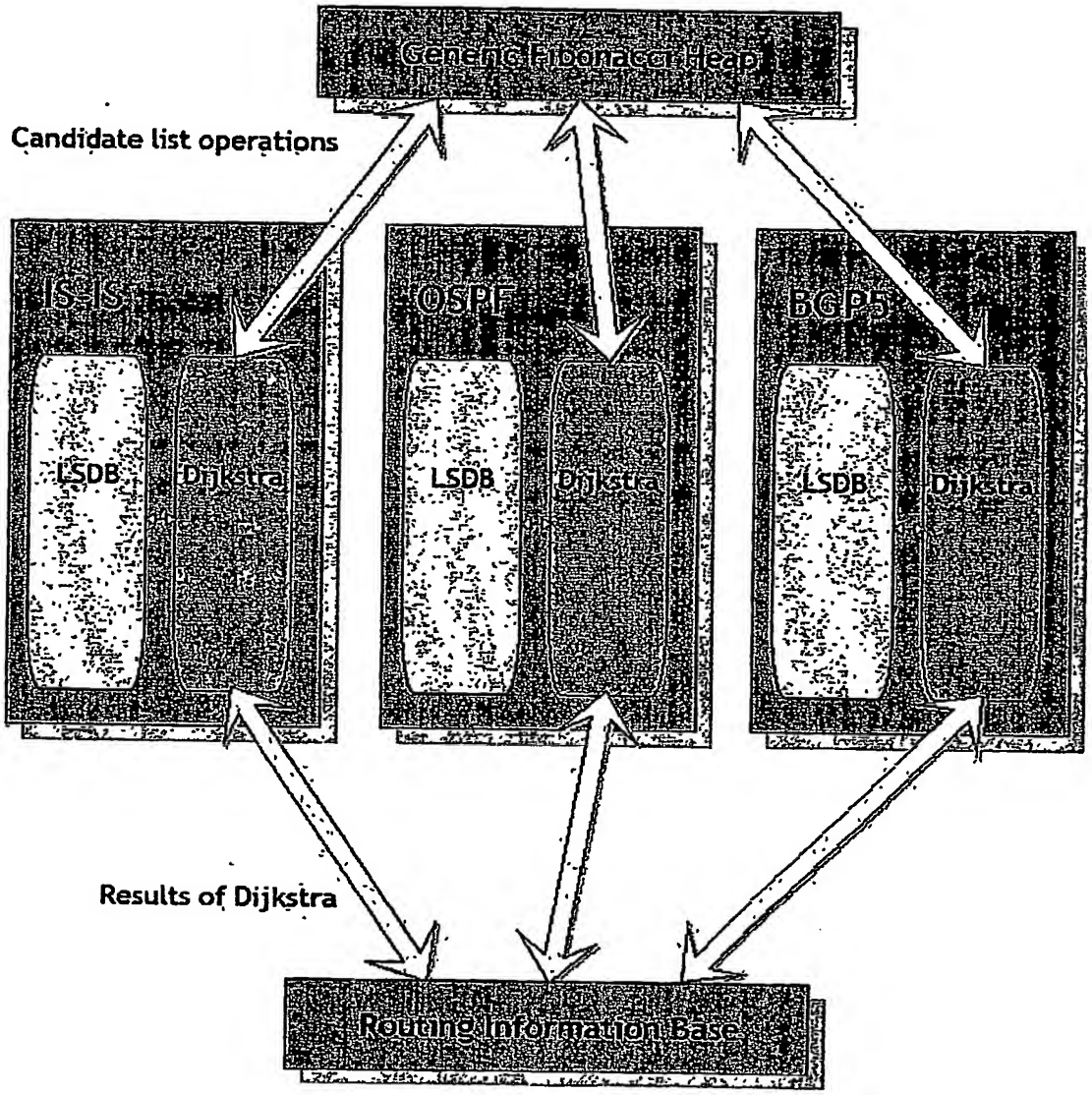


Figure 1

[redacted]

[0023] **Figure 2** illustrates how the shortest path computation interfaces with the generic Fibonacci heap implementation. The computation takes as input a set of vertices discovered through the flooding process, and outputs a set of routes corresponding to the best path(s) through the network.

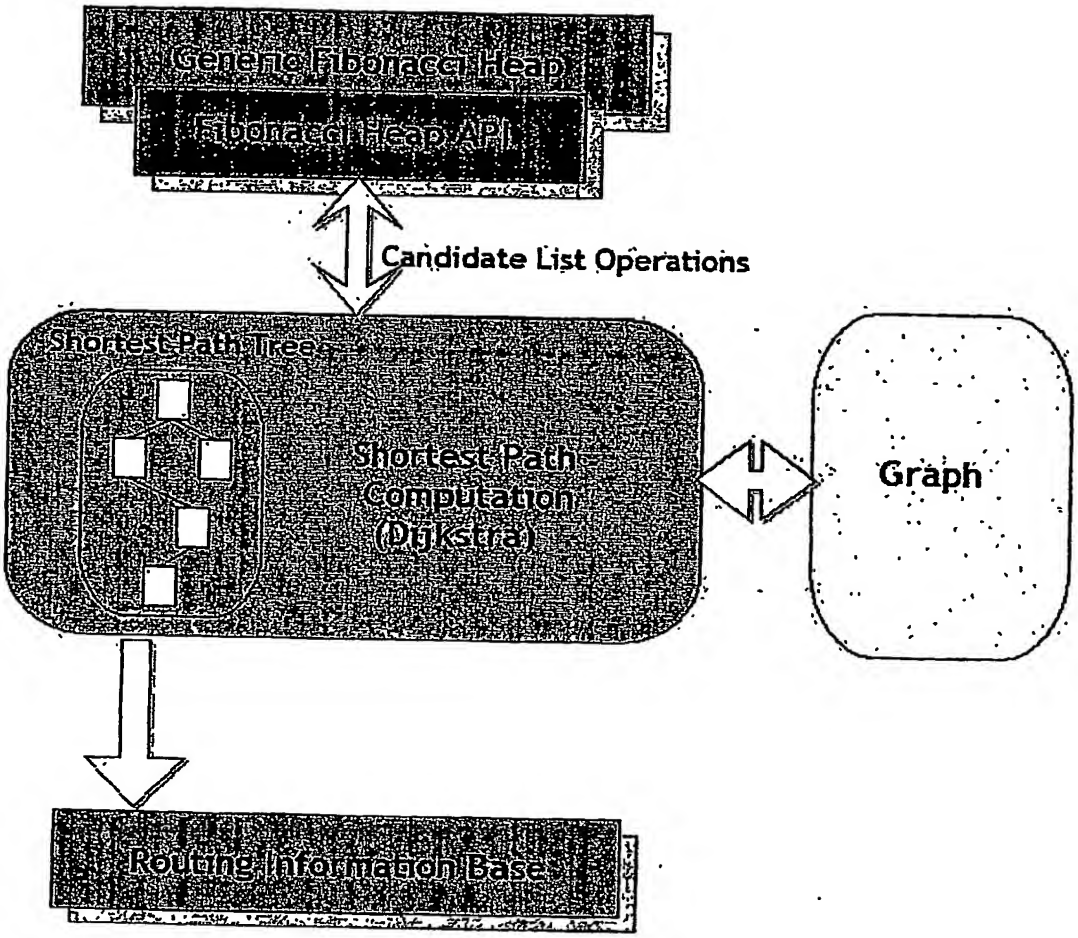


Figure 2

[0024] **Figure 3** illustrates the "generic" element of the implementation. That is, all IS-IS LSPs and OSPF LSAs and nodes are treated as Fibonacci Heap nodes when passed through the API. The heap may serve multiple protocols, while at the same time minimizing complexity.

[redacted]

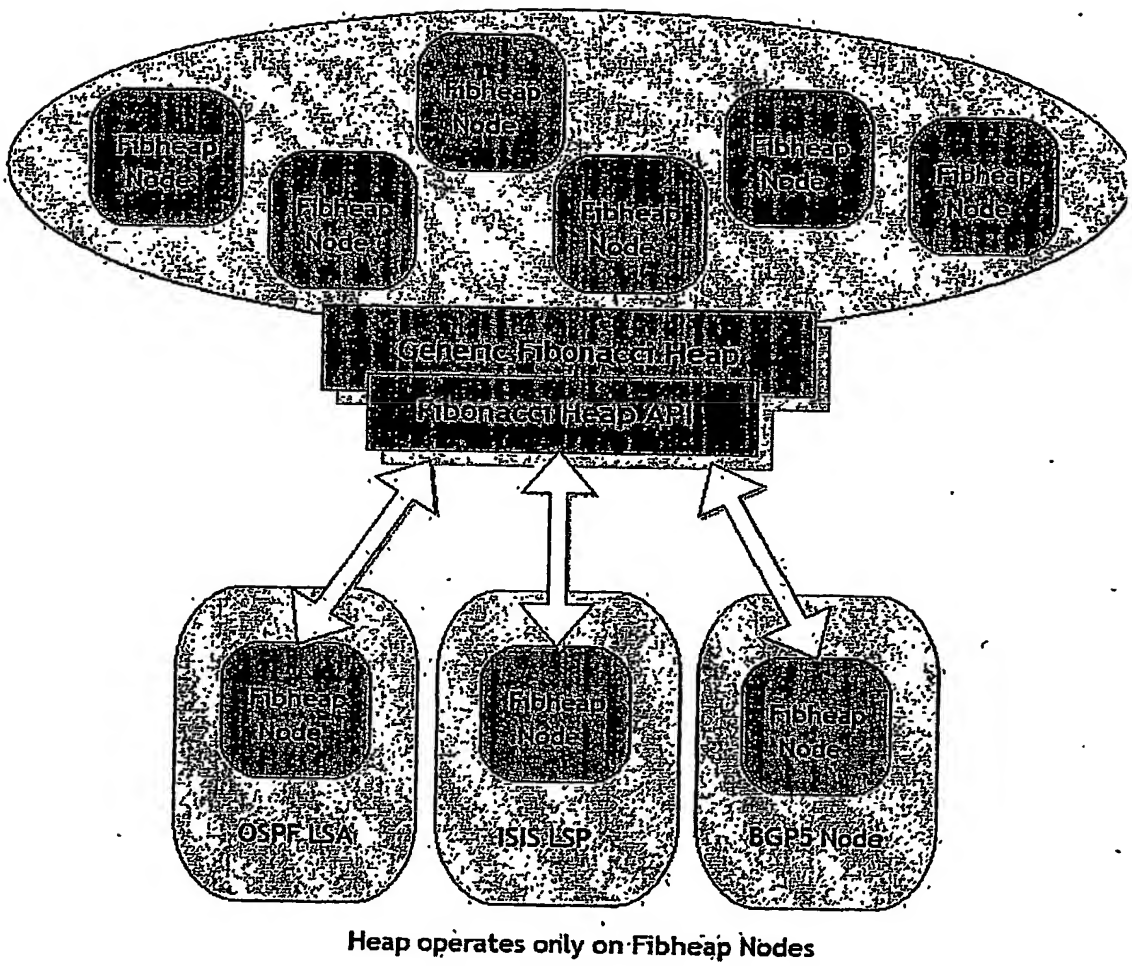


Figure 3

Fibonacci Heap Node

[0025] Each vertex in the graph that has been discovered in the Dijkstra-like computation has an associated candidate list entry, as illustrated in Figure 4.

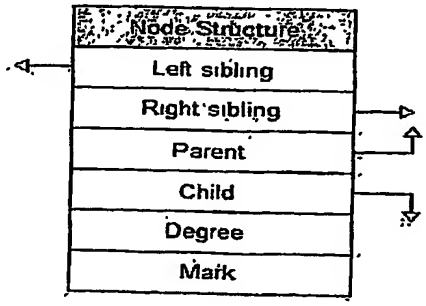


Figure 4, Fibonacci Heap Node

[0026] Each field is further explained in Table 1. Each node is part of a circle queue of siblings and maintains a pointer to its parent.

Field	*Purpose*	*Data Type*
fn_left	Points to the left sibling in the circular queue of siblings of this node	Pointer
fn_right	Points to the right sibling in the circular queue of siblings of this node	Pointer
fn_parent	Points to the parent of this node	Pointer
fn_degree	Set to the number of children of this node	Integer
fn_mark	Set to 1 when a node is made a root, set to 0 when a node loses a child. Node is made root when it is losing a child and its mark is set to 1	Boolean

Table 1

Fibonacci Heap

[0027] The candidate list is represented by the Fibonacci heap top-level structure as illustrated in Figure 5, which represents an instance of a heap:

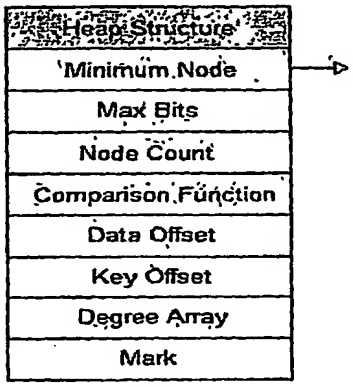


Figure 5, Heap Instance Structure

[0028] One such structure is instantiated per list instance, for example, for each instance of a link state protocol. A pointer to the circle queue of root nodes is maintained, which is the point of access to the list. The purpose of each field is explained in Table 2.

Field	*Purpose*	*Data Type*
f_min	Points to the minimum cost node, also used to access a circle queue of root nodes	Pointer
f_max_key_bits	Set (by API user) to *lg (n)* of the largest key	Integer

Field	*Purpose*	*Data Type*
f_nnodes	Set to the number of nodes in the heap	Integer
f_cmp_func	Set (by API user) to a comparison function to return -1, 0, 1 indicating relationship between two supplied keys	Function pointer
f_degs	Buffer used internally	Pointer

Table 2

[0029] In all operations, moving a node to the root circle queue means setting its **fn_parent** field to 0 in addition to adding it to the queue pointed to by **f_min** of the owning heap.

[0030] The general layout of an instance of the Figure 5 structure is illustrated in Figure 6. Some fields of the instance structure have been omitted for clarity.

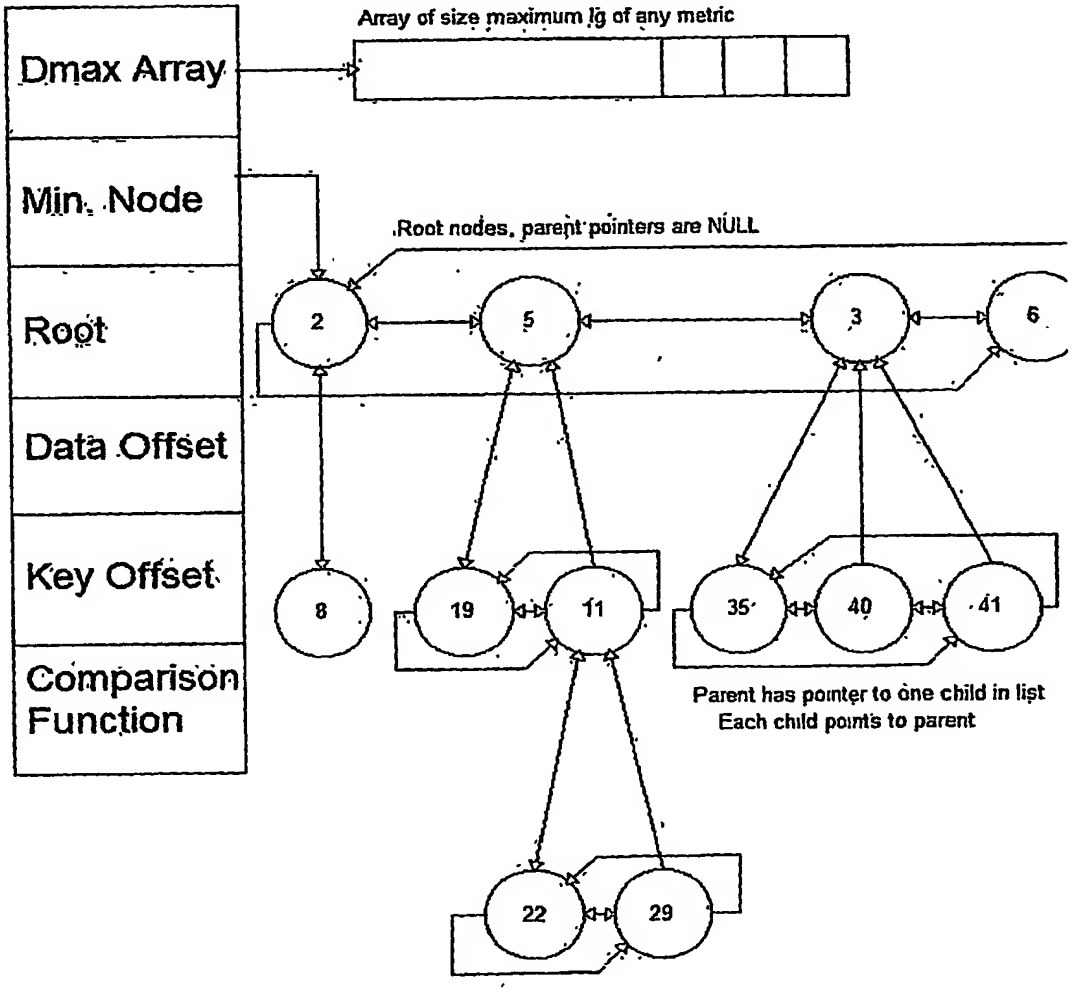


Figure 6

API Description

[0031] This section describes the Application Programming Interface (API) via which the Fibonacci heap structures are manipulated.

Initialization Operation

[0032] *API Definition*

fibheap_init(heap, data_offset, key_offset, maxbits, comparison_function)

Parameters

[0033] **heap** - a pointer to the tree structure that represents an instance of a Fibonacci Heap (see Figure 5).

[0034] **data_offset** - the offset of a pointer in the node that contains [WHAT IS THIS FOR?]

[0035] **key_offset** - the offset of a pointer in the node that contains the key. This parameter may be zero, indicating that the key is an offset into the node itself

[0036] **maxbits** - The maximum number of bits in any key, or the base-2 log of the maximum key.

[0037] **comparison_function** - a pointer to a function used to compare two keys, which should return a value less than, equal to, or greater than zero indicating the relationship between the first and second key.

[0038] *Procedure*

1) The **f_min** and **f_nnodes** fields of the global **fibheap_t** are initialized to zero.
2) The **maxbits** and **comparison_function** are used to initialize these values in the fibheap instance structure.
3) An array of size **maxbits** pointers is allocated and stored in the instance structure.

[0039] The initialization operation is illustrated in Figure 7.

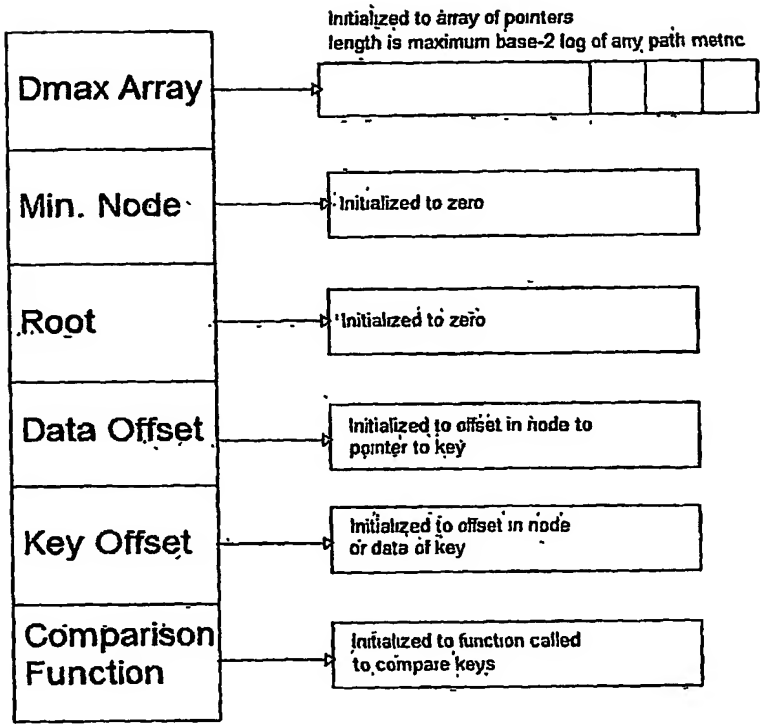


Figure 7

Insert Operation

[0040] *API Definition*

fibheap_insert(tree, node)

[0041] *Parameters*

tree - a pointer to the heap instance structure (see Figure 5).

node - a pointer to a node structure (see Figure 4).

[0042] *Procedure*

1) The new **node** is placed on the circle queue of root nodes, referenced by the minimum node pointer in the heap instance structure (see Figure 5).
2) The user-supplied comparison function is called with arguments **node** and the current minimum node. If this function returns a value less than zero, then the minimum pointer in the heap structure is set to point to **node**. The insert operation is shown in Figure 8.

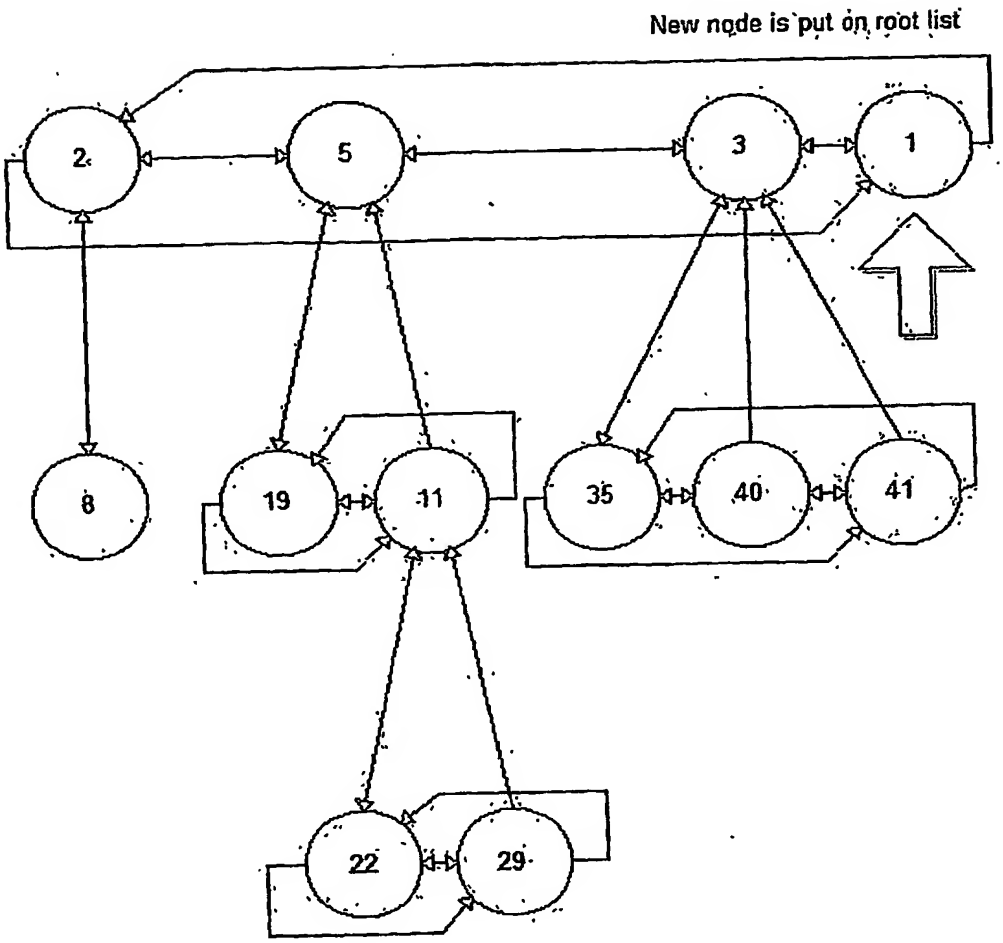


Figure 8

Extract Minimum Operation

[0043] *API Definition*

fibheap_extract(heap);

Returns the minimum node.

[0044] *Parameters*

heap - a pointer to the heap instance structure (see Figure 5).

[0045] *Procedure*

1) The minimum node is removed from the circular queue.
2) All of the children of this node are moved to the root circle queue.

3) The heap is consolidated using the following procedure:
    a. The buffer of size **maxbits** pointers (see Figure 5) is initialized to zero.
    b. The root circle queue is walked, setting the Nth entry in the array to point to a parent if its degree is N.
    c. If there already exists an entry in N for this degree, then the heaps are merged, keeping the heap property (i.e. no child can be greater than its parent).
        i. When this occurs, the new heap of degree N+1 is now referenced by the N+1 entry in the array.
    d. At the end of this procedure, the nodes referenced by the array form the root circle queue.
4) The new minimum is found by walking the root circle queue.
5) The number of nodes is decremented.

[0046] This operation is illustrated in Figures 9-12.

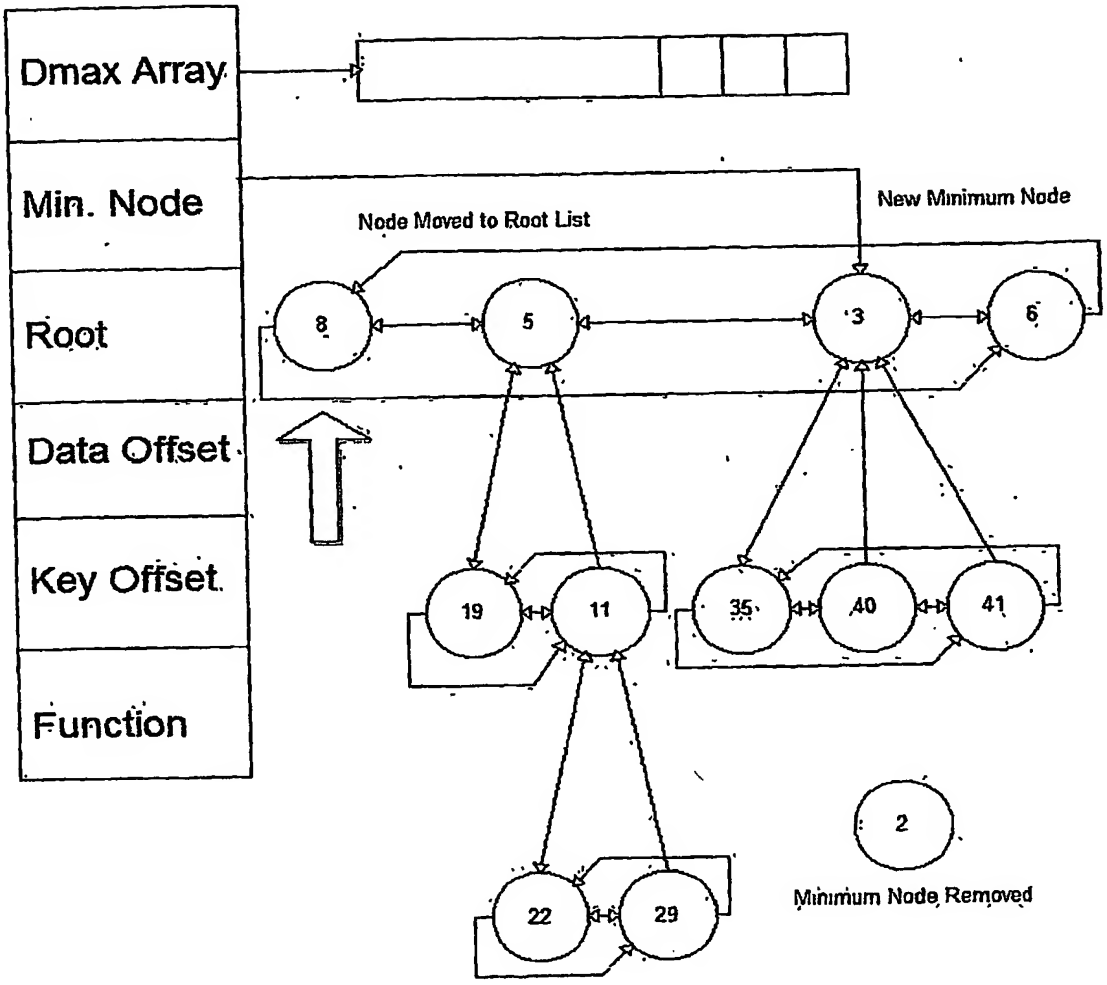

Dmax Array
Min. Node
Root
Data Offset
Key Offset
Function
Node Moved to Root List
New Minimum Node
8
5
3
6
19
11
35
40
41
22
29
2
Minimum Node Removed

Figure 9

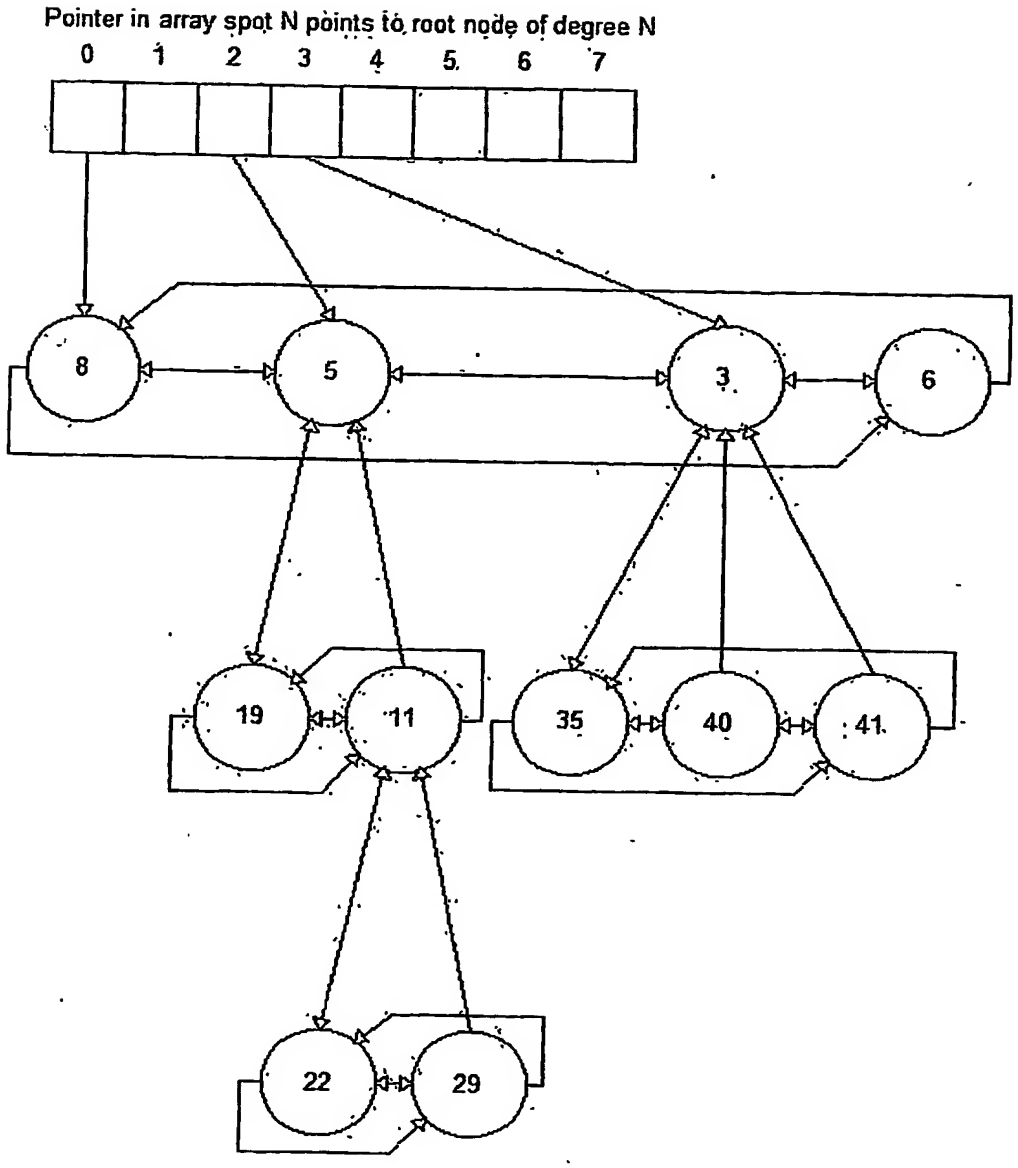
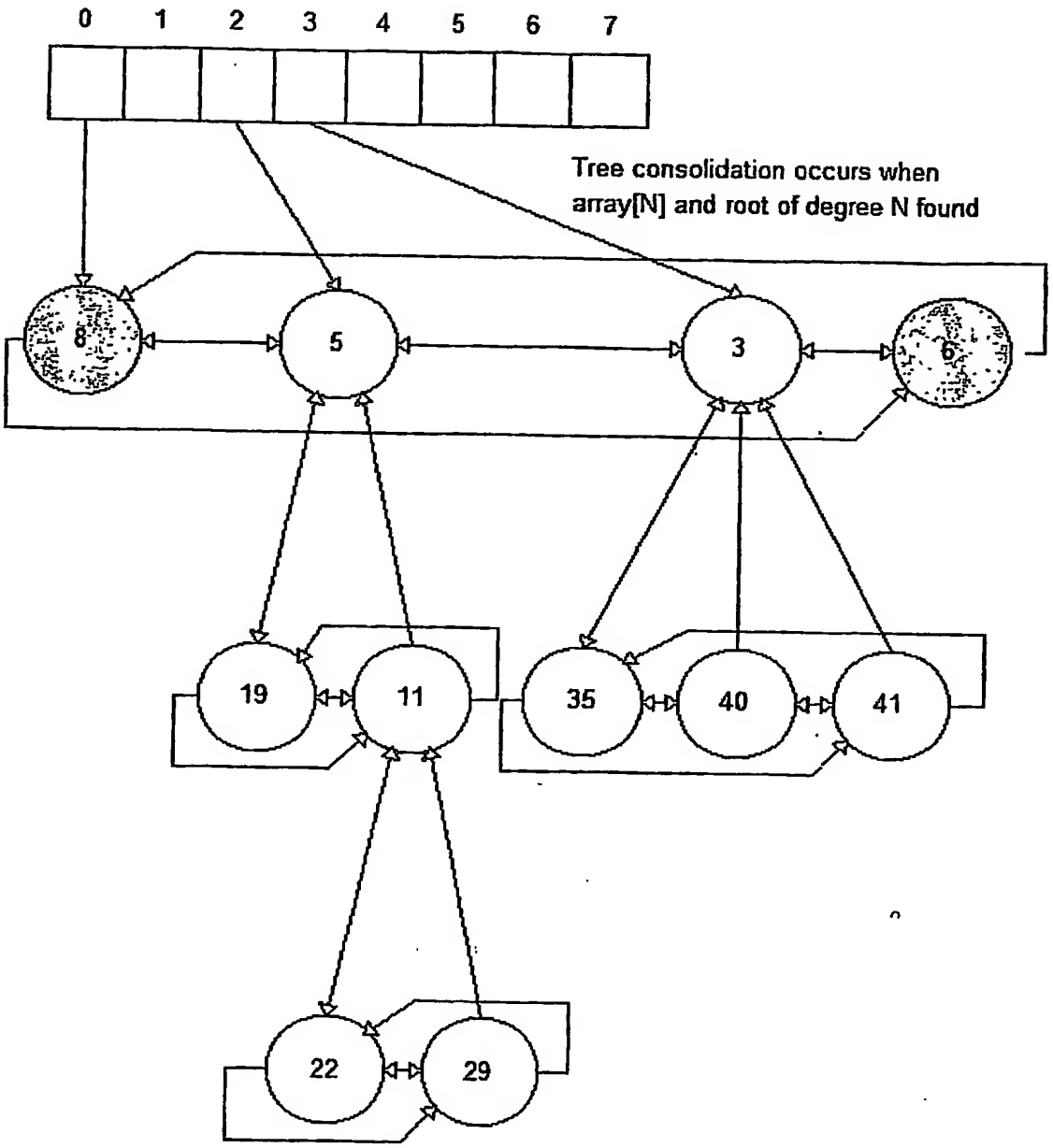

Pointer in array spot N points to root node of degree N
0
1
2
3
4
5
6
7
8
5
3
6
19
11
35
40
41
22
29

Figure 10


0
1
2
3
4
5
6
7
Tree consolidation occurs when
array[N] and root of degree N found
8
5
3
6
19
11
35
40
41
22
29

1.

Figure 11

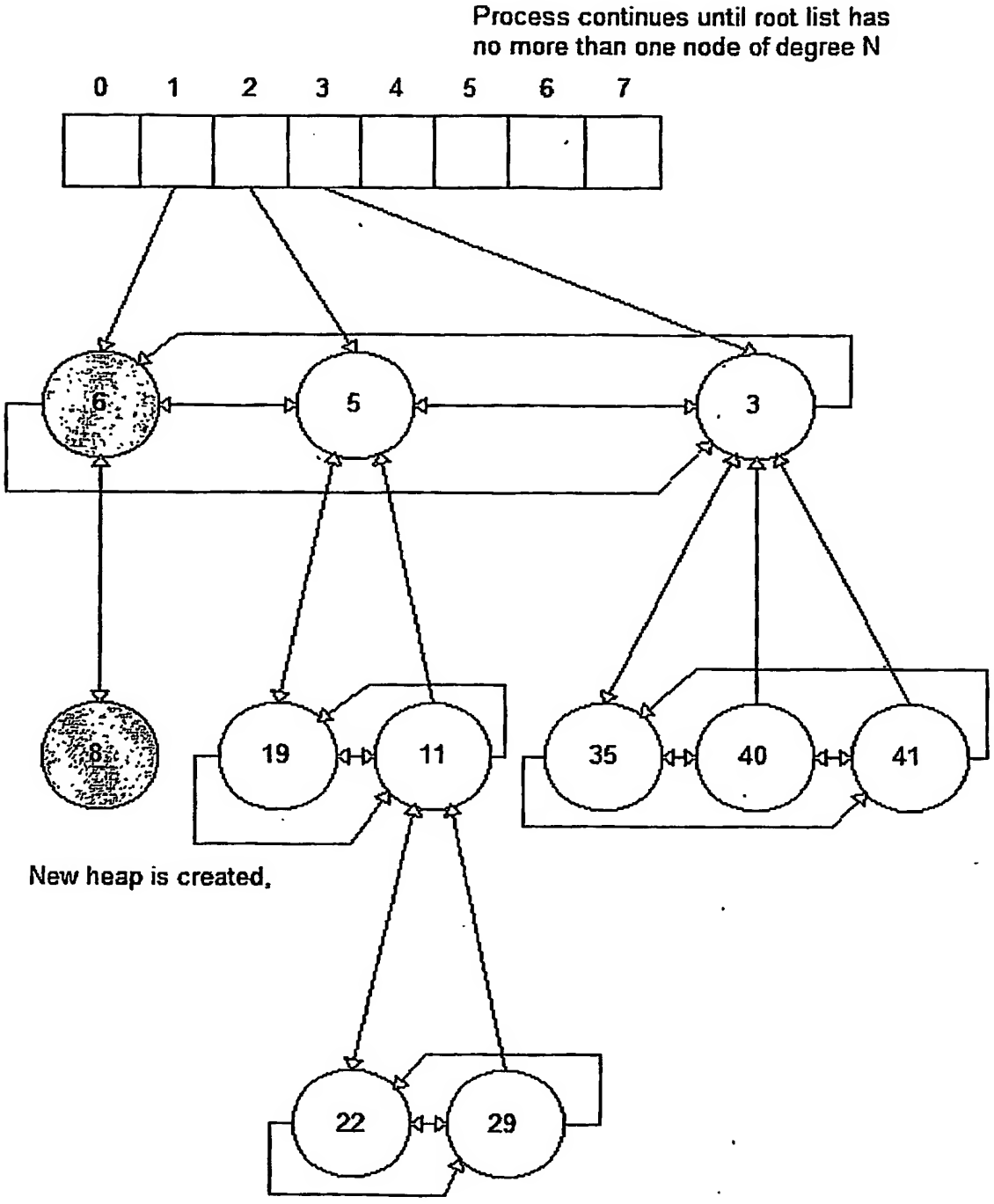

Process continues until root list has no more than one node of degree N
0
1
2
3
4
5
6
7
6
5
3
8
19
11
35
40
41
New heap is created.
22
29

Figure 12

Relax Key Operation

[0047] *API Definition*

fibheap_key_changed(tree, node)

[0048] *Parameters*

tree - a pointer to a heap instance structure (see Figure 5)

node - a pointer to a heap node structure (see Figure 4)

[0049] *Procedure*

1) If the key of **node** is less than the current minimum node, then **node** becomes the minimum node. The comparison is done with the function given in the initialization operation.
2) If **node** was on the root circle queue, then the operation terminates.
3) Else, if the key of **node** is smaller than its parent key, then **node** is moved to the root list.
4) As long as there is a parent available, the following process is repeated. Nodes on the root circle queue do not have a parent pointer.
    a. If this node has lost its first child, then its mark is set.
    b. Else, if the mark is set, then move the node to the root circle queue.
    c. Examine the parent of this node.

[0050] This operation is shown in Figures 13-16.

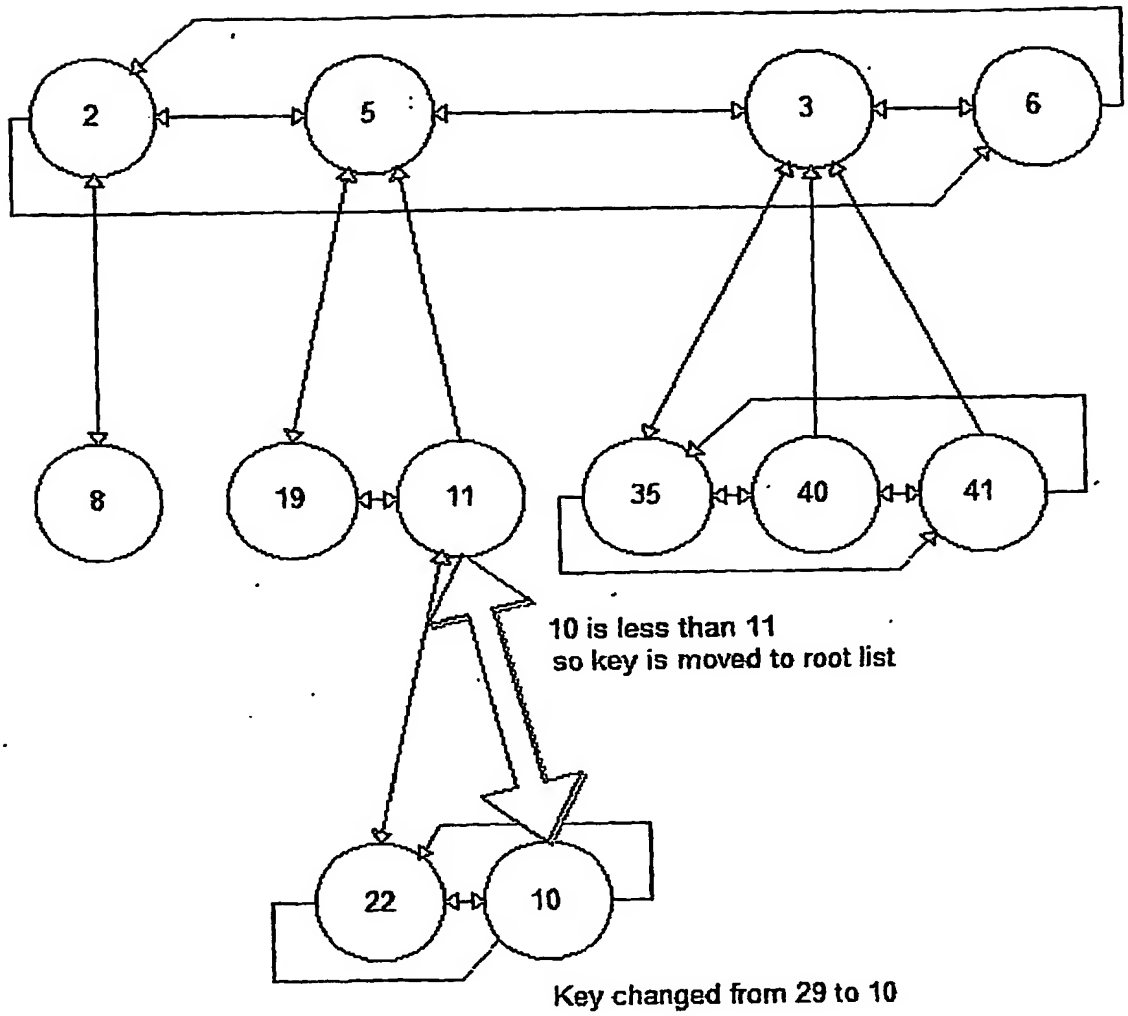

2
5
3
6
8
19
11
35
40
41
10 is less than 11
so key is moved to root list
22
10
Key changed from 29 to 10

Figure 13

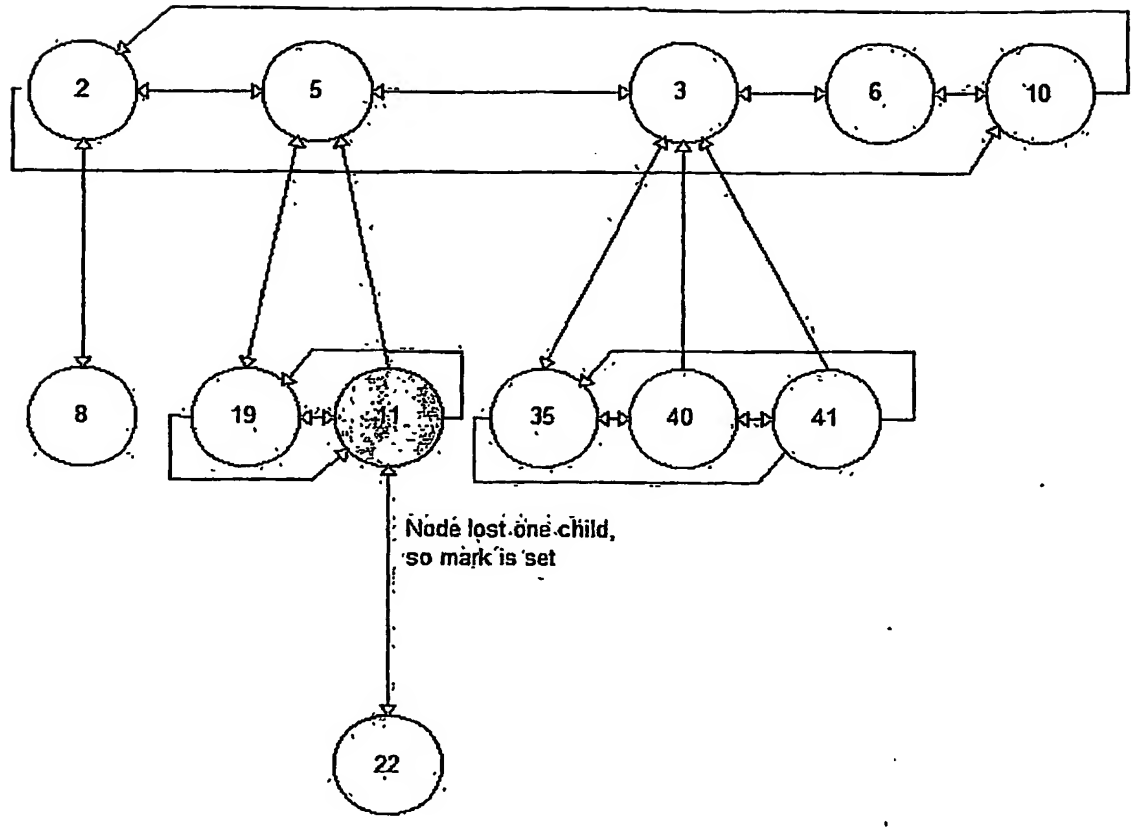
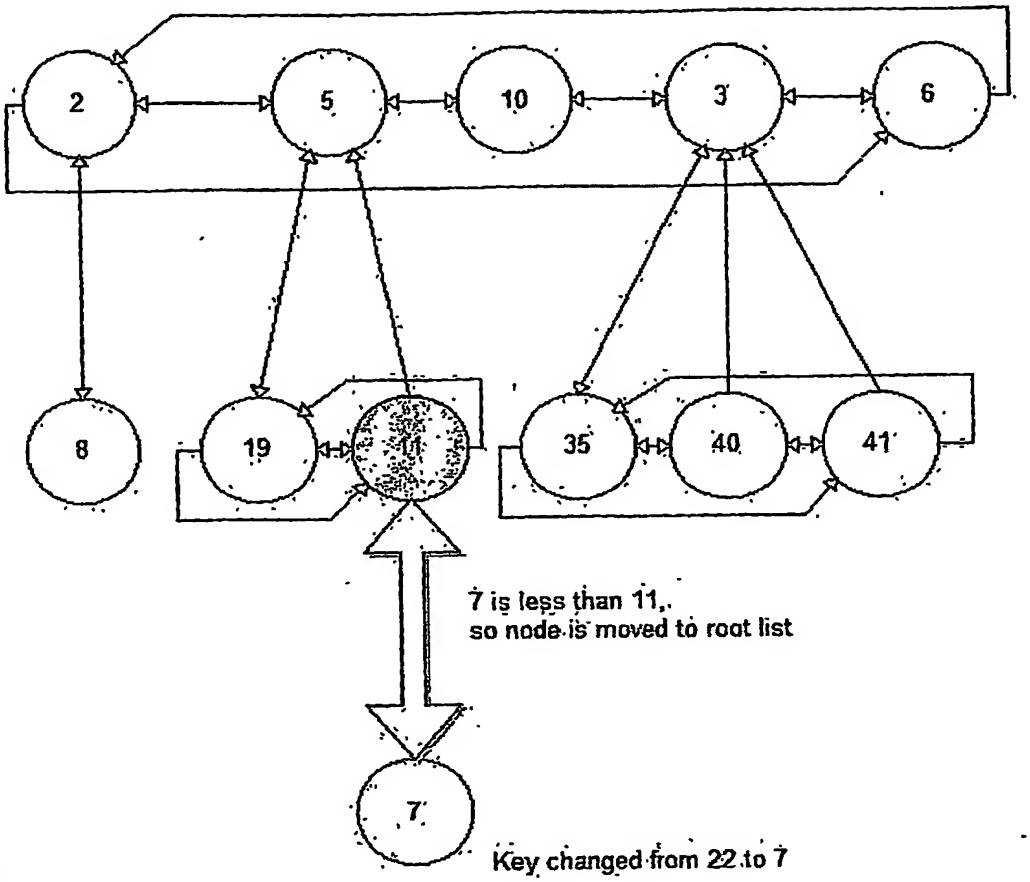
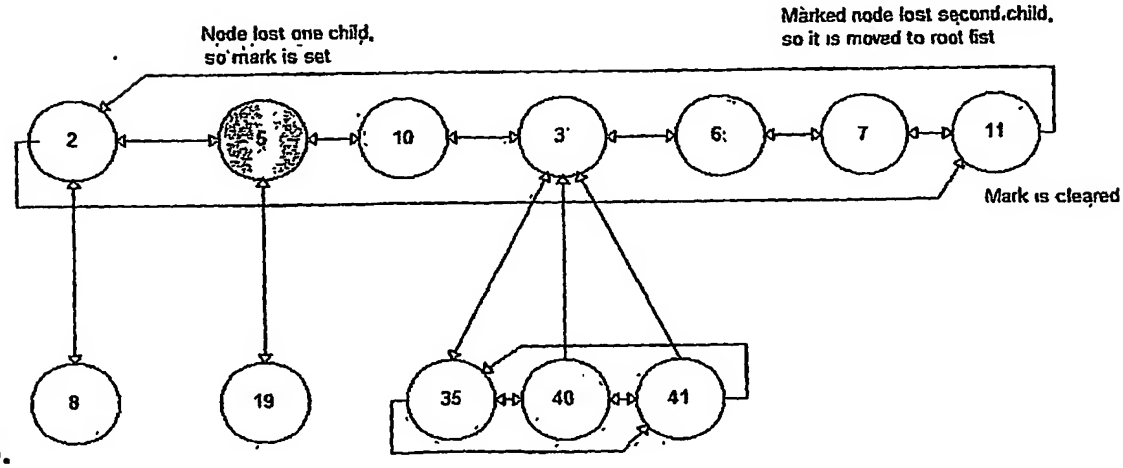


Figure 14



2.

Figure 15



3.

Figure 16

Advantages

[0051] Even though Dijkstra algorithms with Fibonacci heaps have been discussed in mathematical journals, it has been considered that actual implementations of the Fibonacci heaps would be slow and complex. Particular implementation details can be important to achieving speed with minimized complexity. The next sections, 1.6.1 through 1.6.3, describe examples of such implementation details.

The described implementations are fast and have reduced complexity. Not only is IP route computation faster, but these implementations lend themselves to being scalable.

Theoretical vs. Practical Issues with Fibonacci Heaps

[0052] The determination of minimum weight spanning trees is a well-known graph problem. Widely used solutions to this problem include the Bellman-Ford[1], D'Esopo-Pape[2], and Dijkstra[3] algorithms. The Dijkstra algorithm is the basis for the routing computation in Internet link state routing protocols, and is referred to here as the "Dijkstra-like" algorithm because each protocol defines a specialized way of applying the algorithm. Fredman and Tarjan[4] proposed a Fibonacci heap as a new way of storing the candidate list used in the Dijkstra, which improves the algorithm's theoretical worst-case bounds[5] to $O(V \lg V + E)$.

[0053] The theoretical performance of the Fibonacci heap algorithm is promising, but experimental evidence has indicated that Fibonacci heaps are conventionally not useful in practice because they are complicated[6] and slow in practice[7]. This may be summarized as:

---

[1] R. Bellman, "On a routing problem", Q. Appl. Math, vol 16, pp. 87-90, 1958

[2] D. Berksekas, "Linear Network Optimizations: Algorithms and Codes", MA, Cambridge: MIT Press 1991

[3] E. Dijkstra, "A note two problems in connection with graphs", Numerical Math, vol. 1, pp. 269-271, 1959

[4] M. Fredman, R. Tarjan, "Fibonacci Heaps and their uses in improved network optimization algorithms", 1987, ACM 004-5411/87/0700-0596

[5] Cormen, Leiserson, Rivest, *Introduction to Algorithms*, MIT Press 1990, ISBN 0-262-03141-8, p530

6 J. Stasko, J. Vitter, "Pairing Heaps", Experiments and Analysis", p 235 paragraph 2, 1987, Communications of the ACM, Volume 30 number 3

7 Rajeev Raman, "A Summary of shortest-path results", December 1996, p 7

[0054] "From a practical point of view, however, the constant factors and programming complexity of Fibonacci heaps make them less desirable than ordinary binary (or k-ary) heaps for most applications. Thus, Fibonacci heaps are predominantly of theoretical interest. If a much simpler data structure within the same amortized time bounds as Fibonacci heaps were developed, it would be of great practical use of well."[8]

[0055] We have overcome the practical limitations generally described as "programming complexity" and "constant factors" in a specialized domain: the shortest path computation in Internet link state protocols. **The result is a significant performance improvement in the link state routing protocols.** These factors are described in the next sections.

Constant Factors That Have been Addressed

Requirement of Auxiliary Array

[0056] The general Fibonacci heap definition has a requirement of an auxiliary array which must store at least D_{max} pointers to nodes, where D_{max} is equal to the maximum log of the set of keys used.

We allocate the auxiliary array initialization time based on a limited maximum log.

Recursive Definition

[0057] The general definition is recursive. In many environments, recursion is impractical.

We make procedures iterative. For example, the "cut" operation, performed in the *extract-minimum* operation of the generalized API, effectively recursively examines the parent node to see if that node needs to be moved to the root queue. Our implementation sets the parent pointer of the roots on the node list to NULL, meaning that an iterative function may operate on ancestor nodes using a set parent pointer as a loop invariant.

---

8 Cormen, Leiserson, Rivest, Introduction to Algorithms, MIT Press 1990, ISBN 0-262-03141-8, p 420

9 B. Moret and H. Shapiro, "An Empirical Assessment of Algorithms for Constructing a Minimum Spanning Tree", DIMACs in Discrete Mathematics and Theoretical Computer Science, 1991,

Bookkeeping Overhead

[0058] Many bookkeeping fields are required for maintenance of the tree, utilizing extra storage per node, (e.g., for left and right sibling pointers and a parent pointer, along with the "mark" indicator). This is more storage than typically utilized with other data structures.

Other Problems That Have Been Addressed

Programming Complexity

[0059] The "programming complexity" has been reduced and modularized for Internet link state routing protocol domain, so improving the efficiency by which implementation can be utilized.

No Support for Lookup

[0060] The conventional Fibonacci heap does not support lookup based on cost or other key. This is required, for example, in part (2) step (d) of the routing computation in OSPF, as the candidate list entry for a vertex must be retrieved. We address this drawback by keeping a pointer to the candidate list entry structure for a vertex, so it may be retrieved without lookup.

Differing Time Bounds on Operations

[0061] The "extract min" operation may take longer than the *insert* or *relax-key* operations because of the tree consolidation that occurs immediately afterwards. While this may negatively affect some applications, notably those that need some guaranteed bounds on the components of the Dijkstra-like computation, it is not an issue for the domain of link state routing protocols since the computation typically occurs all at once, if even for only a part of the spanning tree.

Integration with OSPF

[0062] This section describes the application of the Fibonacci heap algorithm and data structure to the shortest path computation in OSPF.

Candidate List Representation

[0063] The candidate list referred to by section 16.1 of RFC 2328 is represented by the following simple structure.

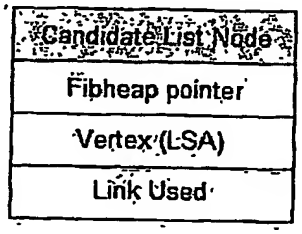


Figure 17

The purpose of each field of this structure is explained in Table 3.

Field	*Purpose*	*Data Type*
l_fnode	Fibonacci heap node	Pointer
l_vtx	Points back to the owning vertex (LSA)	Pointer
l_vl	Points to the incoming link used to reach the owning vertex (LSA)	Pointer

Table 3

[0064] The **vertex_t** structure represents a single OSPF LSA. This structure contains a pointer back to the **cdtlist_t** that represents this LSA in the heap. Since the heap does not support efficient lookup, this pointer provides for increased performance. This structure is the only portion of the LSA representation relevant to this invention. The word *node* in this document is used interchangeably with LSA, meaning each node in the Fibonacci heap represents an LSA encountered in the shortest-path computation.

[0065] The next sections describe the algorithm including definitions of the API in ANSI C. The operations used by OSPF in section 16.1 are initialization, *insert*, *extract-minimum*, and *relax-key*. These are used in the implementation of section 16.1 of RFC 2328 as follows.

[0066] In step 1, *initialization* is used to initialize the data structures used for the candidate list.

[0067] In step 2, part (d), bullet 3, if the cost D is less than the current cost for vertex W on the candidate list or W does not have an entry, insert is used to insert an entry for W on the list of there was no existing entry, or *extract-minimum* is used to decrease the cost of W on the list.

[0068] In step 3, the node in the candidate list with the least cost is chosen. The *extract-minimum* operation is used to extract the vertex in the candidate list with the smallest key.

[0069] In summary, the Fibonacci Heap algorithm is applied to a specific component of the specialized process in OSPF used to calculate IP routes. The use of the algorithm for the optimization of the algorithm in Section 16.1 of RFC 2328 results in dramatic scalability improvements and improved operational performance in an OSPF implementation by reducing the amount of time required to compute IP routes in an OSPF area.

Use of the Fibonacci Heap in IS-IS

[0070] The Intermediate System to Intermediate System (IS-IS) protocol is a *link-state* protocol that uses mechanisms similar to those used in OSPF. The IS-IS protocol is described in ISO Standard 10589.

[0071] Link state information is flooded in the form of LSPs (Link State Packets). IS-IS uses a two-level routing hierarchy, dividing the domain into separate *levels*. The shortest path computation is run independently for level 1 and level 2. The result of these computations are used for the same purpose as OSPF - to maintain forwarding state.

[0072] The algorithm used in IS-IS is described in Appendix C, section C.2.4 of ISO 10589. Two sets of vertices are maintained: TENT and PATHS. The candidate list used in OSPF is loosely analogous to TENT in IS-IS; it contains the set of vertices to which it is not known if the best path has been discovered.

[0073] The set of vertices (LSPs) in TENT is manipulated in the following parts of the algorithm described starting in section C.2.4:

1)In C.2.5 Step 0, the TENT list is initialized to zero.
2)In C.2.6 Step 1, part(d), a vertex may have its key (metric) changed.
3)In C.2.7 Step 2, part (a), the minimum cost vertex is extracted from TENT.
4)In C.2.7 Step 2, part(a), a vertex may be placed into TENT.

[0074] The list operations may be summarized as *initialization, insert, extract-minimum,* and *relax-key.*

[0075] Some benefits of the application of the Fibonacci heap algorithm and data structure to the IS-IS TENT list are:

- The time required to run the algorithm defined in section C.2.4 to completion is insignificantly decreased in the presence of a large IS-IS topology.
- Due to the decreased running time of the computation, the results may generally be computed more often (leading to more accurate forwarding state) or be given less restrictions (such as being allowed to run without interruption).

Integration with IS-IS

[0076] This section describes the application of the Fibonacci heap algorithm and data structure to the shortest path computation in IS-IS.

TENT Entry Representation

[0077] A destination (network or router) discovered in the shortest path computation is an entry in TENT.

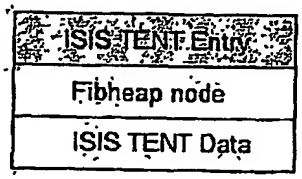


Figure 18

[0078] A subset of the fields is described in Table 4. Fields not described in the table are not relevant to this description.

Field	*Purpose*	*Data Type*
dh_fnode	Fibonacci heap node	Pointer
IS-IS TENT Data	Fields for IS-IS	Various

Table 4

[0079] The **dh_fnode** field maintains the state of the destination with respect to TENT. A global counter is incremented before each iteration of the

shortest path algorithm. When a destination is placed in TENT its **dh_fnode** field is set to the value of the counter.

[0080] A destination represents, among other things not relevant to this description, any type of vertex found in the graph (a network or router). Each vertex has its own Fibonacci heap node represented in the **dh_fnode** field.

[0081] In summary, the Fibonacci Heap algorithm is applied to a specific component of the specialized process in IS-IS used to calculate IP routes. The use of the algorithm for the optimization of the algorithm in Section C.2.4 of ISO 10589 results in dramatic scalability improvements and improved operational performance in an IS-IS implementation by reducing the amount of time required to compute IP routes in an IS-IS level.